Optimal and Approximate Approaches for Selecting Proxy Agents in Mobile Network Backbones

Ahmed Kamal
Dept. of Electrical & Comp. Eng
Iowa State University
Ames, Iowa 50011-3060
kamal@iastate.edu

Hesham El-Rewini*
Department of Computer Science & Engineering
Southern Methodist University Dallas, Texas 75275-0122
rewini@engr.smu.edu

Raza Ul-Mustafa
Dept. of Electrical & Comp. Eng
Iowa State University
Ames, Iowa 50011-3060
raza@iastate.edu

Abstract

In a mobile environment, each mobile host should have a home agent on its home network that maintains a registry of the current location of the mobile host. This registry is normally updated every time a mobile host moves from one subnet to another. We study the tradeoff between the cost of updating the registry and the cost of searching for a mobile host while it is away from home. Using a set of special agents, called *proxy agents*, which implement a two-tier update process, the cost of updates could be reduced; however, the search cost might increase. We study different approaches to identify a set of proxy agents that minimizes the cost of search. In this paper, we use mathematical programming to obtain optimal solutions to the problem. We consider two situations: the cost of search measured by the sum of all search message costs, and the cost of search measured by the maximum cost of such messages. For these two respective cases we formulate the minimization of the cost of search as *Min-Sum* and *Min-Max* problems. For large networks in which the optimization problem may be intractable, we study three different approximate approaches: 1) clustering, 2) genetic algorithms, and 3) simulated annealing. Results of a large set of experiments are presented.

1. Introduction and Related Work

The rapid advances in wireless communication and portable computers have contributed significantly to a recent shift towards a mobile computing paradigm. An increasing number of users continue to move from one location to another carrying portable computing and communication devices. These mobile users need to remain connected to the Internet while they

* El-Rewini's research is supported in part by the DoD Army Research Office Grant # DAAD19-01-1-0399.

are on the move. There is a growing need for mobility support on the Internet. The goal of mobility support is to provide the means by which computers are able to communicate seamlessly even when their points of attachment to the network may have changed. Several approaches for accommodating mobile hosts on the Internet have been proposed and are described in [10]. *Mobile IP* is perhaps the most feasible approach among the various options available to support host mobility. It is an extension of the IP network protocol, which allows mobile computers equipped with wireless network interfaces to communicate with each other, and with computers on the fixed network [11-13].

In a network environment with mobile users, each mobile host belongs to a home network and is assigned a unique home address. All hosts in the network use this home address to communicate with a mobile host regardless of its current location. Each mobile host must have a home agent on its home network that maintains a registry of the binding between the home address and the current location (*care-of address*) of the mobile host. When a mobile host moves to a different network, called foreign network, it can either register with a foreign agent, or acquire a temporary local address. If the mobile host registers with a foreign agent, it uses the unique address of the foreign agent as its care-of address. The foreign agent sends the current care-of address to the home agent, allowing the home agent to update its registry. If the mobile host acquires its own temporary local address, it contacts the home agent directly and provides the home agent of its new care-of address, which then updates its own binding. For the rest of this paper, we will assume that a foreign agent will always be involved in the registration process.

A fixed network can be considered as consisting of a network of subnets, such that each subnet is the home for all hosts with a common IP prefix. Each subnet is assumed to include one mobile agent, which can serve as a home agent for all hosts that belong to its subnet. This mobile agent can also serve as a foreign agent for mobile hosts that although belong to other subnets but are visiting its subnet. Different schemes exist to forward packets to the mobile host. One of the schemes, used in Mobile IP, is described below. When a host sends a packet destined for a mobile host, it is routed to the mobile host's home subnet using standard IP routing. The home agent intercepts all packets intended for the mobile host and then uses the binding information in its registry to forward the packets to its current care-of-address. The foreign agent forwards the original packet to the mobile host.

In the above scheme, the binding at the home agent should be updated every time a mobile host moves from one subnet to another. This of course will give the mobile host's home agent the most up-to-date information regarding the mobile host's current location. Thus, packets intended for a mobile host can be forwarded to its current subnet's agent, which should have no problem locating the mobile host in its subnet if it has not already left. However, updating the binding every time a mobile host moves from one subnet to another could be very costly. In addition to the cost of updating the registry at the home agent, update operations also involve exchange of messages between the mobile host and its home agent through the foreign agent. One optimization goal might be to reduce the number of updates, especially since some updates could actually end up being unnecessary. For example, if a mobile host changes its current subnet before receiving any packets, updating the binding at its home agent becomes useless and unnecessary in this case. The number of updates can be reduced if a mobile host does not have to update its home agent's registry every time it moves from one subnet to another. Instead, updates at the home agent are preformed only when a mobile host moves from one neighborhood to another. A *neighborhood* is defined as a set of neighboring subnets. In a scheme like this, one mobile agent in each neighborhood should be identified to act as a *proxy agent* on behalf of all mobile agents in the neighborhood. (It is to be noted that in the context of GPRS/UMTS, this proxy agent is known as a *Gateway Foreign Agent* (GFA), and acts as a proxy to a number of *Foreign Agents* [14]. Assignment of FAs to GFAs is a static problem, and does not change with time.)

When a mobile host moves from one neighborhood to another, the mobile host requests services from the mobile agent of the subnet it has just visited. This mobile agent will then inform its neighborhood's proxy agent of the arrival of the mobile host. The proxy agent will in turn register its address at the mobile host's home agent as the care-of address. This is the only case in which the binding at the home is updated as long as the mobile host stays in this neighborhood. When a mobile host moves from one subnet to another within the same neighborhood, the mobile host only requests routing services from the mobile agent of the subnet it has moved to, and no update will take place at the home agent. When a home agent intercepts packets destined to a mobile node's home address, it will forward them to the proxy agent of the neighborhood in which the mobile host exists. Unlike previous scheme, now the proxy agent has to search for the mobile host among all the subnets that belong to its neighborhood. Clearly, this

scheme introduces the cost of searching a neighborhood for a mobile host. Thus there is a trade off between the cost incurred by updating a binding registry and that incurred by searching for a mobile host in a neighborhood. If the binding registry is updated every time a mobile host moves to a new subnet, no searching will be required to find the new location. On the other hand, some searching for the mobile host will be required if the binding registry is updated only when the neighborhood is changed. It should be also noted that the search could be totally avoided if we allow mobile hosts to perform local updates to the proxy agent with every subnet move. In this case, a global update is reduced to a local update. However, this can result in a large number of local updates if mobile hosts change subnets frequently.

The problem we address in this paper is how to identify a set of proxy agents in order to achieve some balance between update and search costs. The cost of search can be computed in one of two ways:

1. As the sum of all the individual search costs from the proxy agent to all the mobile agents in its neighborhood
2. As the maximum search cost between the proxy agent and its mobile agents.

These two cases may, for example, correspond to the two cases in which the search can be done by querying the mobile agents in the neighborhood of the proxy agent one by one, or by querying the agents by sending a broadcast message, respectively. Or, as another example, by considering the cost of search to be the bandwidth requirement and the maximum delay, respectively.

Many researchers, as in [1, 2, 5], have studied this two-tier approach. Our contribution in this paper is the criteria and methodologies of optimally choosing the proxy agents. In searching for an optimal selection of the proxy agent according to the above two criteria, we take a mathematical programming approach, more precisely, an integer linear programming approach [6]. For large networks the optimization problem may be intractable. Therefore, for such cases, we also study three different approximate approaches: 1) clustering, 2) genetic algorithms, and 3) simulated annealing. We present the results of many experiments conducted to study the effectiveness of using these approaches. We also show the CPU time needed to solve different instances of the problem.

Many researchers are exploring the tradeoff between the cost of search and the cost of update, in different network settings. Considerable attention has been given to modeling, computing and optimizing the cost of location management, e.g. in [8,9,15,16]. The idea of designating a subset of network nodes to perform update operations has been utilized to reduce the cost of update. For example, Bar-Noy and Kesler [2] studied a related idea to minimize the cost of search and update in cellular architectures. They identify a subset of all base stations and designate them as reporting centers. Mobile users transmit update messages only upon entering cells of reporting centers. They introduced sequential optimal and near optimal algorithms to find reporting centers in some restricted cases. However, they only considered special graphs. In [7], the authors show that the problem is equivalent to either p-center or p-median finding problems in graph theory. They introduce a distributed strategy to solve the general problem and introduce optimal distributed algorithm for special cases. The proxy agent approach was also recently introduced in the Session Initiation Protocol (SIP) [5] that employed a location server. The location server basically acts similar to a proxy agent. However, in [5], the issue of selecting such servers was not introduced. Another related work by Badrinath *et al.* [1] examines strategies to reduce the search cost and at the same time control the volume of location updates by employing user profiles. Also, in the context of ad hoc networks, the idea of proxy agents has been used for routing. Solutions to the problem of finding the *minimum connected or independent dominating set,* in graph theory, have been used to determine a subset of nodes that can serve as proxy agents [3, 4, 16].

The rest of the paper is organized as follows. In Section 2, we present the model we used to represent the problem of finding proxy agents in fixed networks with mobile users. In Section 3, we introduce our mathematical programming approach. In Section 4, we present three approximate approaches, namely clustering heuristic, genetic algorithms, and simulated annealing. Experimental results are summarized in Section 5. We finally give our concluding remarks and directions for future work in Section 6.

2. Modeling the Problem

In this section, we model the fixed network backbone using an undirected weighted graph, which we call the backbone graph. We also define strategies to calculate the cost of search and the cost of update.

2.1 Backbone Graph

A mobile computing environment consists of a set of subnets. Each subnet has one mobile agent that can serve as a home agent for hosts that belong to this subnet. It can also serve as a foreign agent for mobile hosts that although belong to other subnets but are visiting its subnet. A set of neighboring subnets is called a neighborhood. One of the mobile agents in each neighborhood is identified as the proxy agent for this neighborhood. All other mobile agents will be referred to as mobile agents.

We represent the above architectural model using backbone graph, which is an undirected, connected, weighted graph $G = (V, E)$, V is a set of vertices and E is a set of edges. Each vertex in the backbone graph represents a subnet. Since each subnet has exactly one mobile agent, each vertex in the graph also represents a mobile agent. Two vertices are connected by an edge if and only if the two subnets are connected by a direct link. Associated with each edge is a weight that reflects the communication cost over the link represented by that edge. Figure 1 shows an example of a backbone graph. In the figure, we show seven subnets represented by seven agents. The weights of the edges represent the communication cost between the mobile agents of the related subnets, and we assumed that this cost is symmetric[1]. For example, the communication cost between the mobile agents of the subnets c and e is 4.

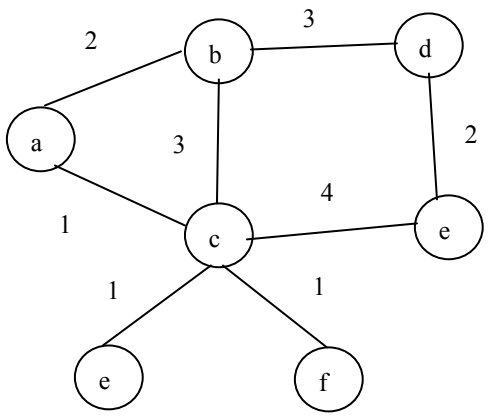

Figure 1: Backbone Graph representing 7 subnets and their interconnection. The weight on the edges represents the communication cost between the mobile agents of the related subnets.

2.2 Cost of Search

The cost of search could change depending on the strategy by which the search is conducted. It can be argued that the two ways to compute the search cost, as discussed in Section 1, are equivalent to the cost of conducting a depth-first-search and a breadth-first-search, respectively. If the search is conducted using a depth first strategy, the cost of searching could be calculated as the summation of the communication cost between every node and its nearest proxy agent. On the other hand, if the search were conducted using a breadth first strategy, the cost of searching would be the largest communication cost from a node to its nearest proxy agent. In what follows, we formally define the above two strategies to calculate the cost of searching. Given backbone graph $G = (V, E)$, let us assume the following:

$d(i,j)$	Communication cost along the shortest path between nodes i and j in G
P	Number of proxy agents
U_p	Set of proxy agents, $U_p \subset V$
u_i	A proxy agent that is a member of U_p.

The cost of search can be summarized as follows:

- The cost between a node v and its nearest proxy agent = $minimum_i\ (d(v,u_i))$, $1 \leq i \leq p$
- The cost of depth first search = $\sum minimum_i\ \{d(v,u_i)\}$, $1 \leq i \leq p$, $\forall\ v\ \varepsilon\ V$.
- The cost of breadth first search = $maximum\ \{minimum_i\ \{d(v,u_i)\}$, $1 \leq i \leq p\}\ \forall\ v\ \varepsilon\ V$.

The problem is to find the set of proxy agents U_p that minimizes the cost of search as defined above. We shall refer to these optimization problems as *Min-Sum* and *Min-Max* in the depth-first and breadth first search cases, respectively. It can be shown that *Min-Sum* Problem is equivalent to finding U_p that minimizes $\sum\ \{d(v,u_i)\}$, $1 \leq i \leq p$, $\forall\ v\ \varepsilon\ V$. As an example, when $p = 2$, we find the optimal solution of the Min-Sum and Min-Max problems for the backbone graph shown in Figure 1. As shown in Figure 2, $U_p = \{c, d\}$ is the solution of the Min-Sum problem, i.e. mobile agents c and d are the selected proxy agents. The search cost in this case is 8. Similarly, Figure 3 shows that the optimal solution of the Min-Max problem is $U_p = \{a, d\}$ with search cost = 2. The neighborhood of each proxy agents is also shown in the figures.

[1] Asymmetric bi-directional links can also be considered within this framework.

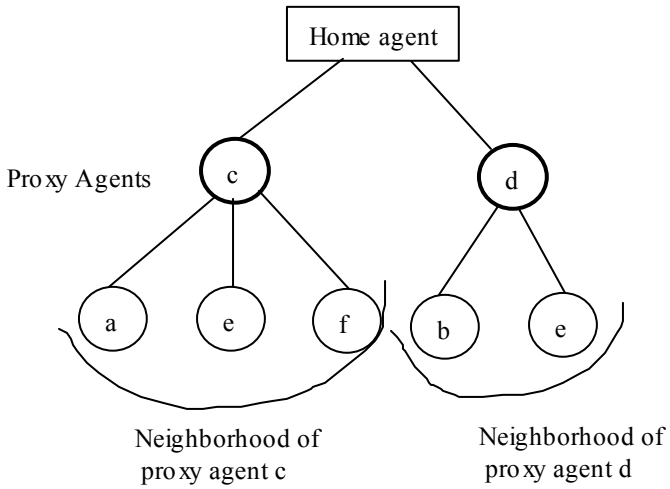

Figure 2: An optimal Solution to the Min-Sum Problem for the Backbone Graph of Figure 1 (P=2). Mobile agents of subnets *c* and *d* are the selected proxy agents that minimize the cost of the depth first search. Hence $U_p = \{c, d\}$.

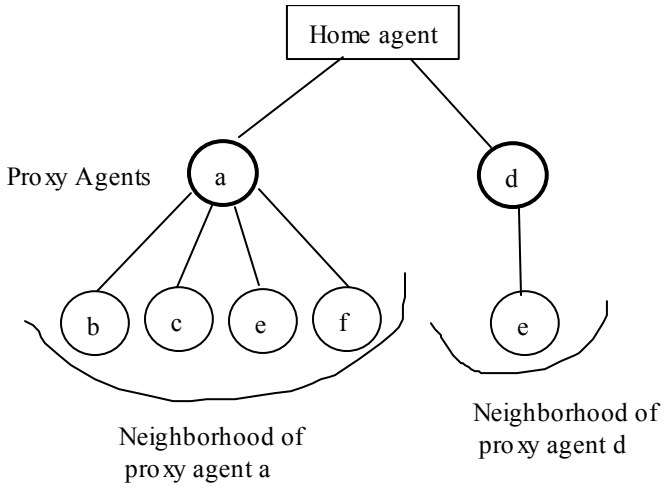

Figure 3: An optimal Solution to the Min-Max Problem for the Backbone Graph of Figure 1 (P=2). Mobile agents of subnets *a* and *d* are the selected proxy agents that minimize the cost of the breath first search. Hence $U_p = \{a, d\}$.

2.3 Cost of Update

Since updating the home agent's registry is performed only when a mobile host moves from one neighborhood to another, the cost of update should be function of the number of proxy

agents. This is because each proxy agent performs the update on behalf of all the agents in its neighborhood (i.e. number of neighborhoods = number of proxy agents).

Having determined p proxy agents, the cost of updating can be modeled as follows. Let us define the parameter k, which we call mobility factor. It is an indication of the degree of mobility. The update cost can be defined as $(ap^k + b)$, where a and b are parameters that reflect the network performance. Note that if k is zero, there is no mobility, while if k is close to 0.5[2], there is a moderate level of mobility. High mobility can be modeled when k is much greater than one, which corresponds to random movement.

3. Mathematical Programming Approach

In this section, we introduce a binary linear programming approach to solving the *Min-Sum* problem, and another integer linear program to solve the *Min-Max* problem. Given backbone graph $G = (V, E)$, we define the following:

V	Set of mobile agents (nodes in the graph G), $\|V\| = N$.
D_{ij}	Communication cost along the shortest path between agents i and j. (d_{jj} is always 0)[3]
p	Number of proxy agents
U_p	Set of proxy agents, $U_p \subset V$, with $\|U_p\| = p$.
u_i	A proxy agent that is a member of U_p.
Y_{ij}	A binary indicator which is 1 if and only if agent i uses agent j as a proxy agent
Z	An integer variable, which corresponds to the maximum among all the minimum communication costs.

[2] The rationale behind the 0.5 factor is due to two assumptions: 1) the distribution of the proxy agents is uniform over the coverage area, and 2) the mobile movement is not random, but directed towards a certain destination. With the first assumption, the proxy agents can be distributed over a square with $\sqrt{p}$ proxy agents per side. The movement of the mobile is therefore a combination of horizontal, vertical or diagonal steps, all of which are functions of $p^{0.5}$.

[3] Without loss of generality, we have assumed that the weights are integers. The case of continuous non-integer weights is a trivial modification.

3.1 Min-Sum Problem

In this section, we try to find the set of proxy agents U_p that minimizes the cost of search when conducted using depth-first strategy. We formulate the following Binary Linear Program (BLP), which can be solved using the branch-and-bound algorithm.

Minimize: $\sum_i \sum_j d_{ij} y_{ij}$

(Note that the $\sum_j d_{ij} y_{ij}$ is the same as $d(v_i, v_j)$, where v_j is a proxy agent as shown in Section 2)

Subject to the following constraints:

1. y_{ij} is integer, and $0 \leq y_{ij} \leq 1 \ \forall \ i,j$
2. $\sum_j y_{ij} = 1 \ \forall \ i$
3. $\sum_j y_{jj} = p$
4. $y_{ij} - y_{jj} \leq 0 \ \forall i,j$

The first constraint guarantees that y_{ij} is a binary variable while the second guarantees that agent u_i uses one super agent only. Constraint 3 guarantees that there are exactly p super agents. Constraint 4 guarantees that agent u_i will use an agent that has been selected as a proxy agent, i.e., an agent u_j for which $y_{jj} = 1$. This constraint is based on the fact that $y_{jj} = 1$ is a necessary condition for $y_{ij} = 1$, i.e. $(y_{ij} = 1) \rightarrow (y_{jj} = 1)$

3.2 Min-Max Problem

In this section, we try to find the set of proxy agents U_p that minimizes the cost of searching when conducted using breadth-first strategy. This problem will be solved in two steps:

1. The first step is similar to the *Min-Sum* problem. We define proxy agents through $\sum_j y_{jj} = p$, and have a node i use proxy agent j by setting $y_{ij} = 1$ only if $y_{jj} = 1$. Note that the y_{ij}'s will not be the same as in the *Min-Sum* problem, since the objective function is different.
2. The second step is to define the variable z, which is greater than or equal to any of the communication costs. The integer linear program minimizes the value of this variable.

Based on the above, we formulate the following Integer Linear Program (ILP), which can be solved using the branch-and-bound algorithm.

Minimize: z

Subject to

1. y_{ij} is integer, and $0 \leq y_{ij} \leq 1 \forall\ i,j$
2. $\sum_j y_{ij} = 1\ \forall i$
3. $\sum_j y_{jj} = \text{p}$
4. $y_{ij} - y_{jj} \leq 0\ \forall\ i, j$
5. z is an integer
6. $z\ \geq \sum_j y_{ij} d_{ij} \forall\ i$

The first four constraints are identical to those of the *Min-Sum* problem and their explanation is straightforward. Constraint 5 guarantees that the variable z, used to determine the maximum among the minima is an integer. In constraint 6, z must be greater than or equal to any of the chosen costs of communication.

Through experimentation, it was found that the same minimal value of the objective function, z, could be obtained with an even smaller value of p. Therefore, the above ILP was slightly modified in order to take this situation into consideration, i.e., to find the minimal value of z such that the number of proxy agents does not exceed a certain upper limit:

Minimize: $\boldsymbol{M_1 z + M_2 p}$

Subject to the following constraints:

1. y_{ij} is integer, and $0 \leq y_{ij} \leq 1 \forall\ i,j$
2. $\sum_j y_{ij} = 1\ \forall i$
3. $\sum_j y_{jj} = p$
4. $y_{ij} - y_{jj} \leq 0\ \forall\ i, j$
5. z is an integer
6. $z\ \geq \sum_j y_{ij} d_{ij} \forall\ i$
7. $p \leq p_{max}$

M_1 and M_2 in the above objective function are positive number, such that $M_1 >> M_2$ which assigns more weight to z as compared to p. Reasonable values for these constants are N and 1, respectively. As such, the ILP attempts to minimize z before attempting to minimize p. p_{max} is the upper limit on the number of proxy agents.

3.3 The Min-P Problem

In this section we consider a different version of the problem, namely, the situation in which the maximum search cost is already determined, and the network designer is faced with the mission of determining the minimum number of proxy agents that will satisfy such constraints. Such a case may prevail when the performance of the network is paramount, and the task of selecting the proxy agents follows next. We therefore formulate the following ILP:

Minimize**: p**

Subject to the following constraints:

1. y_{ij} is integer, and $0 \le y_{ij} \le 1 \forall\ i,j$
2. $\sum_j y_{ij} = 1\ \forall i$
3. $\sum_j y_{jj} = \mathrm{p}$
4. $y_{ij} - y_{jj} \le 0\ \forall\ i, j$
5. $\Sigma_j y_{ij} d_{ij} \le z_{max}$
6. $\sum_i \sum_j d_{ij} y_{ij} \le x_{max}$

Constraints 5 and 6 establish the upper bounds, z_{max} and x_{max}, on the cost of search based on the depth-first and the breadth-first methods, respectively. This is based on the assumption that both methods of search may be employed. However, if either method is otherwise not used, then the upper limit can be set to a very large value in order to exclude such a constraint from the optimization problem.

4. Approximate Approaches

Due to the intractability of this optimization problem, seeking optimal solutions for large networks might not be practical. In this section, we present a number of approximate schemes to solve the large size problems. These approximate techniques are: 1) a simple clustering heuristic, 2) genetic algorithms, and 3) simulated annealing.

4.1 Clustering Heuristic

The idea is to partition the backbone graph into smaller sub-graphs using a clustering mechanism. These small graphs can be solved separately using the techniques given above. Moreover, since these smaller problems are inherently semi-independent, they can be solved in

parallel using a parallel machine or a cluster of workstations. The clustering heuristic presented here is a greedy algorithm and can be described as follows:

Given a backbone graph, do

1. Select the node with the minimum node degree.
2. Add directly connected nodes starting with the closest nodes.
3. Repeat for all the nodes.

In case this method results in disconnected graphs, and some nodes are not added to any of the clusters, a random method of replacement is used such that the new clusters will still be connected. Obviously, this simple clustering heuristic does not attempt to optimize the assignment of nodes to clusters. Our goal of using clustering in this paper is to show the effect of using clustering on the quality of the solution and the execution time. However, in future work it might be useful to also study the effect of different clustering techniques.

4.2 The Genetic Algorithms Approach

Genetic Algorithms (GA) [18] is a general combinatorial search technique. It consists of the following major steps.

1. *Generation*: Generate an initial population of chromosomes
2. *Evaluation*: Evaluate the cost of each individual chromosome
3. *Selection:* Determine the fitness of each individual chromosome
4. *Reproduction*: Reproduce based on fitness, giving more chances to fit individuals to reproduce. In general, use crossover and mutation operators for reproduction.
5. Go to 2, until Stopping criterion is met

There are a number of decisions that one has to make while applying GA to the problem in hand, e.g., representation of the chromosome, mode of initial population generation, mode of preferring the fitter solution, application of crossover and mutation operators and finally, selection of an effective stopping criterion.

We represent a chromosome by a binary string. The length of the binary string is equal to the total number of nodes. All the nodes have a value of binary ‘0’ except proxy nodes, which have a

binary value ‘1’. Figure 4, shows a typical chromosome with nodes 1, 3 and 6 selected as proxy nodes. The vector U_p for the solution shown in Figure 4 is given by {1, 3, 6}.

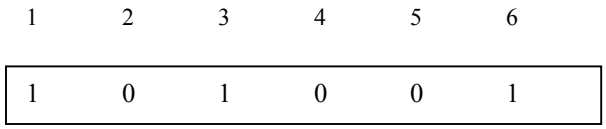

Figure 4: Representation of chromosome in the form of a binary string, for n=6 and p=3. All the nodes have a value of binary ‘0’ except proxy nodes, which have a binary value ‘1’.

We generate the initial population randomly. For *selection* we use roulette wheel. This gives more chance to fitter solutions to be selected for reproduction. The stopping criterion can be either some fixed number of generations or some specific consecutive number of generations for which solution does not improve. We experimented with both stopping criteria, and found the later as more effective than the former in terms of CPU time.

We apply the crossover operator with some probability *pxover*, and select a single point crossover scheme. We randomly select a crossover site and then exchange the part of the chromosomes on right side of the crossover site. Notice however, this may result in infeasible solutions, because the number of proxy agents, here represented by the ‘1’, may become greater or lesser than *p*. Therefore, we modify the exchanging mechanism in such a way that for a chromosome ‘A’ the binary values are copied from right to left of its partner chromosome ‘B’ until the number of ‘1’ in chromosome ‘A’ equals the number of proxy agents. This mechanism will always generates the feasible solutions, and more importantly tends to propagate the most part of the fitter solutions to the next generations. Figure 5 shows how we are applying the crossover operator.

Figure 5: Crossover operation; (a) before crossover (b) after crossover. The part of chromosomes is swapped at the crossover site in such a way to generate a new pair of feasible chromosomes.

Mutation operator is applied with a probability *pmutate* such that *pmutate* is far less than *pxover (pmutate << pxover*). Mutation operator always operates on an individual chromosome. To obtain a new mutated chromosome, we randomly pick two sites in the chromosome such that one of them has a value '1' and another '0', and then just swap their locations. Figure 6 depicts the operation of mutation.

(a) (b)

Figure 6: Mutation operation; (a) before Mutation (b) after Mutation. Binary values at two randomly selected positions that have a value '1' and '0' are swapped. In figure the positions are 2 and 4.

In Figure 7, we show a possible scenario of a complete generation before and after reproduction. We will discuss the experimental results in section 5.3.

Figure 7: Illustration of current generation, and the next generation obtained through the application of crossover and mutation on the current generation.

4.3 The Simulated Annealing Approach

Simulated annealing (SA) [19] is also a general combinatorial search technique. It consists of the following major steps.

1. Generate an initial solution, evaluates its cost, and stores it as the least cost found so far.
2. Get a new solution by searching the neighborhood of the current solution
3. Evaluate the cost of the new solution
4. If the cost is lesser than the least cost found so far, accept it (downhill move)

5. If the cost of the new solution is lesser than the least cost found so far, still accept this new inferior solution but with some probability (uphill move, to avoid getting trapped into a local minimum)
6. Decrease the probability of acceptance of inferior solutions (based on a temperature schedule) after every iteration (to help in convergence)
7. Repeat step 2 through 6, till the temperature approaches zero.

Let S_0 = initial schedule (solution), and the best solution found so far

S_c = candidate schedule,

S_k = current schedule

$N(S_k)$ = neighborhood of the current schedule

β_k = Current temperature, β_f = final temperature (temperature approaching to zero)

α = Cooling rate for temperature schedule

$P(S_{k+1},S_c)$ = Probability of acceptance of new solution (S_{k+1}) given current solution (S_c), and is calculated as follows:

$$P(S_{k+1}, S_c) = e^{\frac{G(S_k)-G(S_c)}{\beta_k}}$$

Then formally the SA algorithm is given as:

Step 1:

Set $k = 1$ and select the initial temperature β_1.

Select an initial schedule S_1 and set $S_0 = S_1$.

Step 2:

Select a candidate schedule S_c from $N(S_k)$.

If $G(S_c) < G(S_0)$, set $S_0 = S_{k+1} = S_c$ and go to Step 3.

If $G(S_c) \geq G(S_0)$, generate $U_k \sim$ Uniform(0,1).

If $U_k \leq P(S_k, S_c)$, set $S_{k+1} = S_c$; otherwise set $S_{k+1} = S_k$; go to Step 3.

Step 3:

Select $\beta_{k+1} = \alpha * \beta_k$

If ($\beta_{k+1} \leq \beta_f$) STOP; Else set $k=k+1$and go to Step 2.

Like GA, we need to take some decisions to be able to use SA for our problem, though the number of decisions is fewer than GA. We select the same binary string representation as used in GA to represent a solution. We define the $N(S_k)$ (i.e., the neighborhood of the current schedule) as all the solutions that can be obtained through pair-wise interchange. Figure 8, shows an example for the pair-wise interchange.

(a) (b)

Figure 8: Generating a new solution from current solution through pair-wise interchange; (a) current solution, (b) new solution.

5. Experiment Results

In this section we will discuss the experimental results for all the solution techniques that we discussed in previous sections, which includes mathematical, Genetic algorithm (GA) and Simulated Annealing (SA). All the experiments were conducted on an SGI Indy workstation.

5.1 Optimal Solutions

We applied the algorithms of Section 3 to several types of networks in order to assess the cost of search, as well as the algorithms execution times. The cost of search, whether it is depth-first or breadth-first, depends on a number of parameters, viz., the number of nodes, the density (the ratio between the number of edges and the number of nodes) and type of the graph, and of course the number of proxy agents. We have conducted several experiments for different values of the above parameters, but always choosing a randomly generated graph. However, it was made sure that the generated graph was connected. The weights of the edges are themselves randomly generated such that they assume integer values, uniformly, from 1 to 10, inclusive. We considered three types of graph densities, R, namely, $O(1)$, $O(\log_2 N)$ and $O(N)$, which correspond to most popular types of networks. At the same time, we considered network sizes of 5, 10, 15, 20, and 25 nodes, where each node corresponds to a mobile agent. The parameter P, which corresponds to the number of proxy agents, was also varied. It assumed values of 1, 2, and 5. In order to guarantee the accuracy of the plotted results, each point in the curves presented in this section is the average of readings obtained from 30 different random graphs. The 90%

confidence intervals of these measurements were also computed and in most cases they were very tight.

The results for the *Min-Sum* and *Min-Max* algorithms are shown in Figures 8 and 9, and are categorized according to the value of *P* in order to study the effect of the number of proxy agents on performance. In the case of Min-Max, we used the modified formulation, $M_1 z + M_2 p$, with M_1 very large, while M_2=1. The *P* values are the upper bounds on *P*, but in so many cases smaller values have been achieved. For example, with N =15 and the R = O(), and upper bound on P was 5, the same objective function was found to be achievable with P=2.

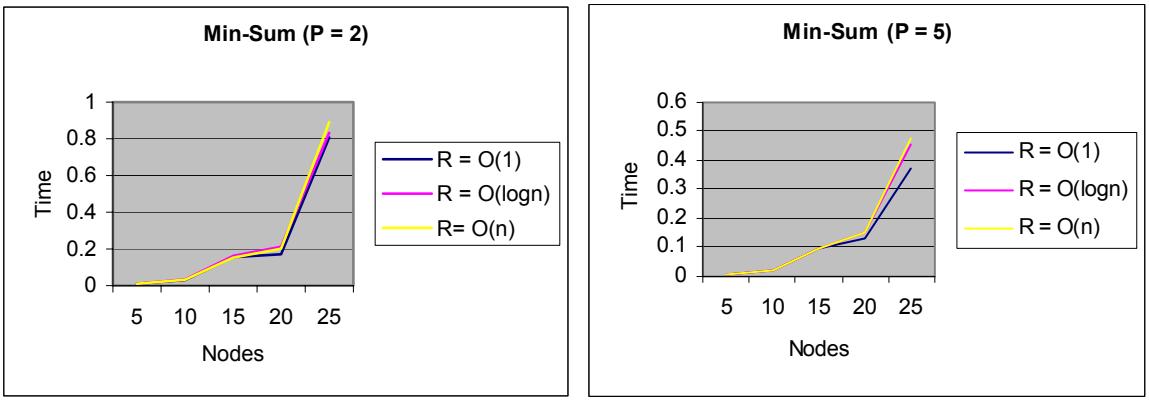

Figure 8: Execution Times when optimal technique is used to solve the Min-Sum problem. Graphs of different densities corresponding to *O(1)*, $O(\log_2 N)$ and *O(N)* were used as an input.

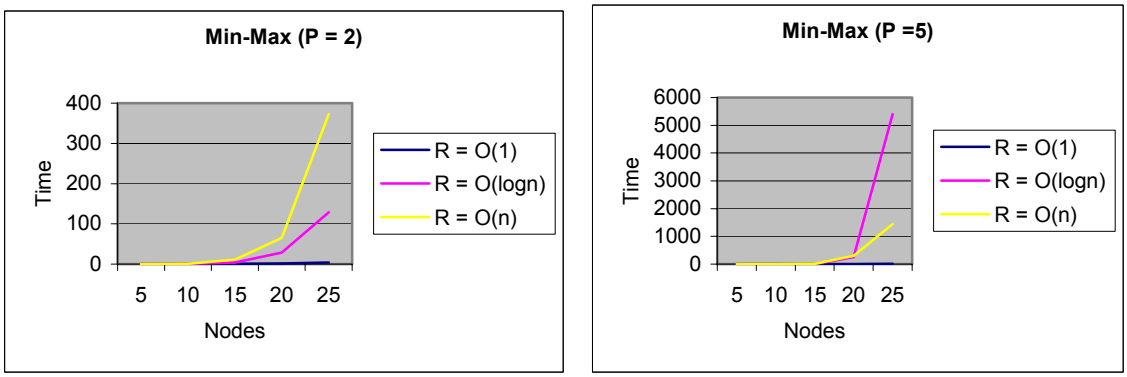

Figure 9: Execution Times when optimal technique is used to solve the Min-Max problem. Modified formulation of Min-Max is used, i.e. $M_1 z + M_2 p$, with M_1 is very large, while M_2=1.

It was observed that generally the execution time is reasonable when the number of nodes in the backbone graph is not too large. It was also shown that the time to find an optimal solution in the case of Min-Sum is consistently less than that in the Min-Max case. Obviously, the objective

function is simpler in the Min-Sum case. Increasing the value of p has always led to an increase in the execution time.

Figures 10 and 11 summarize the results of experimenting with the Min-P problem. Figure 10, shows the value of the objective function (P) averaged over 30 runs, and the execution time. The upper bounds on the sum of costs was chosen to be dependent on the density, and was chosen as 1.5N for $O(1)$, $2N$ for $O(\log N)$ and 2.5n for $O(N)$. The upper bounds on the maximum over all the costs was also chosen to be dependent on both N and the graph density, and was chosen as wN/3 for $O(1)$, wN/5 for $O(\log N)$ and wN/8 for $O(N)$, where w is the average weight per edge. The choice of these bounds was based on the observation that as the graph density increases, smaller costs can be achieved. It should be noted that the value of P is also the average over all 30 runs for 30 different random graphs. However, depending on the graph, the actual value of P can be very much different, i.e., the actual value of P depends very much on the graph. For example, for $N = 25$ and an $O(1)$ density, although the average P was 9.83, P actually varied between 6 and 13. Figure 11, shows the same results except that we assigned greater upper bounds to the maximum cost, and the sum of costs for lower graph densities.

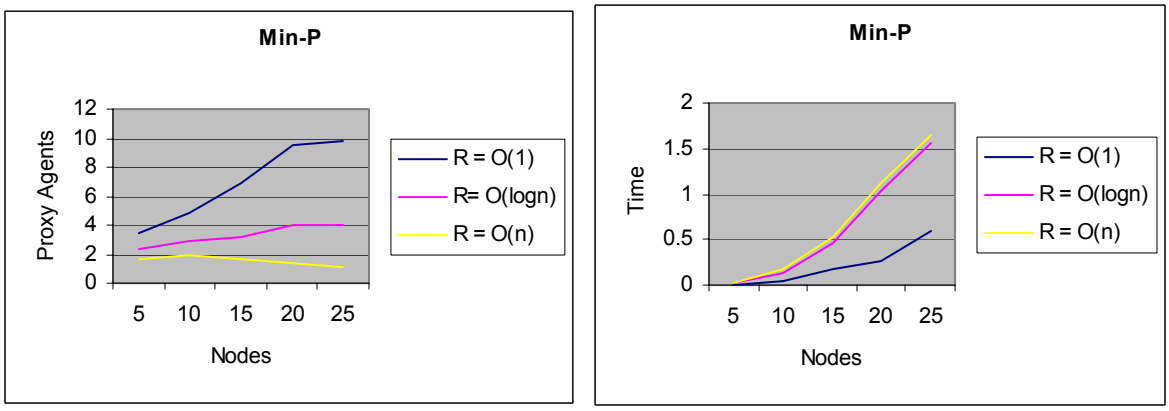

Figure 10: Number of proxy agents obtained and the corresponding execution time while solving Minimum P problem.

5.2 Clustering Heuristic

We also conducted experiments to assess the clustering heuristic presented in Section 3. The heuristic was applied to a large network, which is divided into several *semi-independent* sub-graphs. We chose a network with 100 nodes, and a graph density of $O(\log N)$. We experimented

with 10 and 20 proxy agents. The number of clusters equals 1, 2, 5, and 10, with number of nodes per cluster equals 100, 50, 20, and 10, respectively.

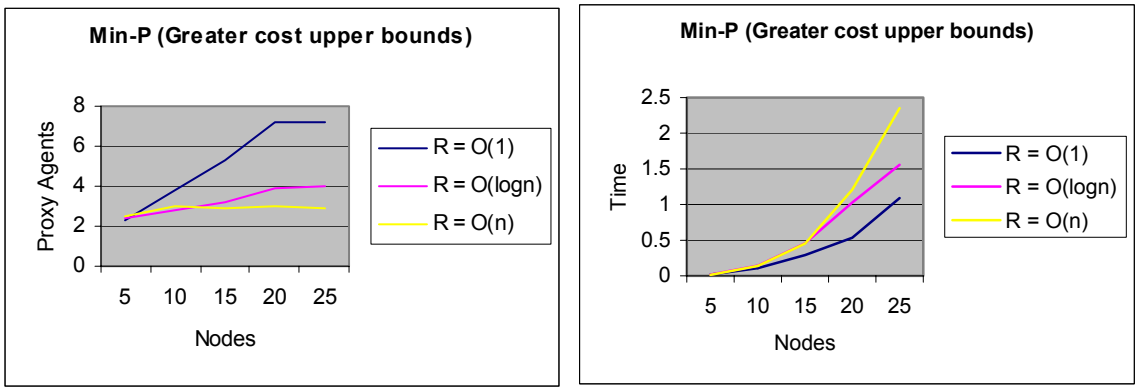

Figure 11: MinimumP problem with greater cost upper bounds.

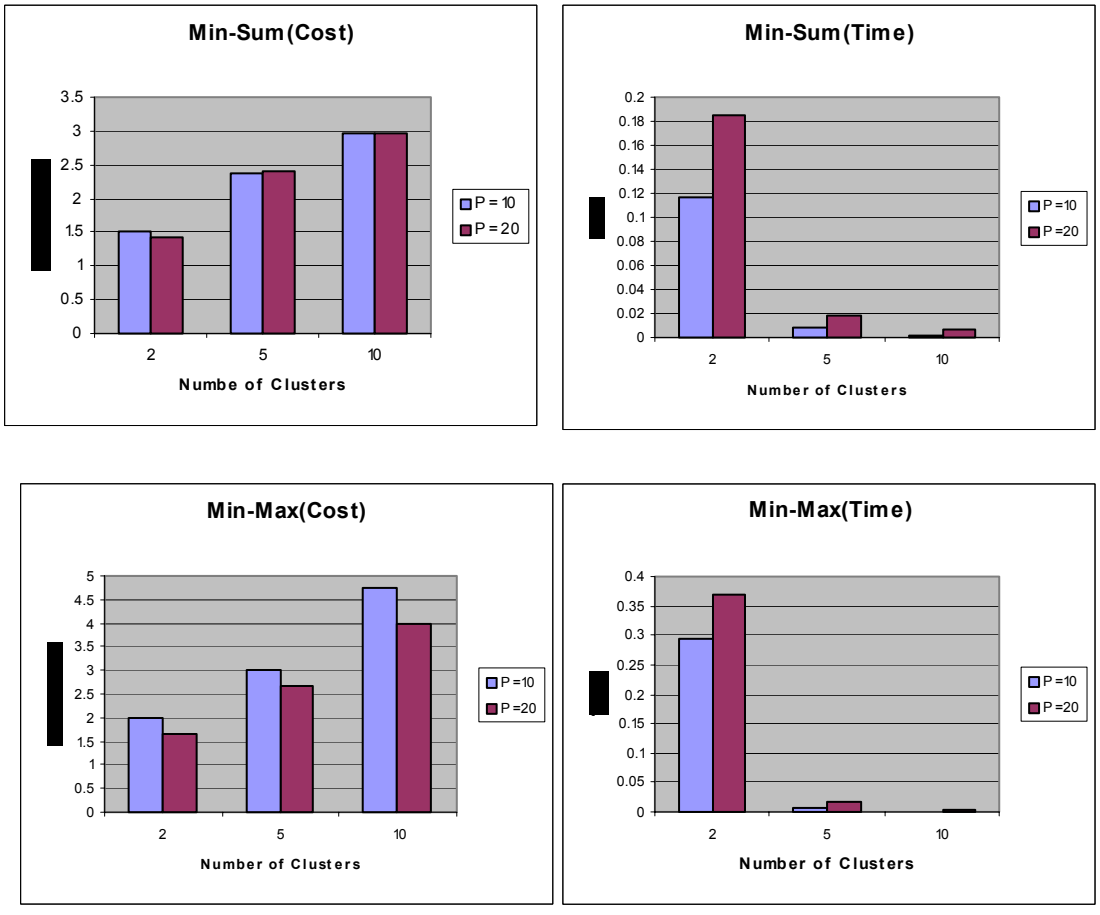

Figure 12: Relative Execution Time and Relative Search Cost of Clustering technique for different numbers of Clusters for P=10, 20.

The relative cost of search and relative execution times for both algorithms is shown in Figure 12. Relative cost is calculated as the cost obtained using the clustering heuristic divided by the optimal cost.

Similarly, the relative execution time is the execution time of the clustering heuristic divided by the execution time of the optimal algorithm. Clearly the optimal search cost obtained when no clustering is used takes a lot more time than that when clustering is used. As the number of cluster increase, the execution time decreases almost exponentially, while the search cost increases close to linearly.

5.3 Genetic Algorithms and Simulated Annealing

In this section we present the experimental results of the Genetic Algorithms (GA) and the Simulated Annealing (SA) technique. We investigated the following question. What is the cost and time trade-off offered by the GA and SA techniques as compared to optimal solution technique? In other words, what speedup is offered by GA and SA and at how much degradation in the solution quality? Moreover, we also investigated the effect of varying P on the relative performance of each technique.

Most of the experimental settings are same as that of the section 5.1. For example, we are again considering the three types of graph densities, $R = O(1)$, $R = O(\log_2 N)$ and $R = O(N)$. Also all the results are collected with two different values of P (P = 2 and P = 5). This time, however, graphs are generated for number of nodes equal to 10, 20, 30, 40 and 50. Results are collected for both the *Min-Sum* and *Min-Max* objectives. In the case of Min-Max, we used the modified formulation, i.e. $M_1 z + M_2 p$, with M_1 is very large, while M_2=1.

To compare the performance of the GA and SA with respect to optimal technique, we have calculated *relative cost* and *relative time*. Relative cost is calculated as the cost obtained using the GA or SA technique divided by the optimal cost. Similarly, the relative execution time is the execution time of the GA or SA technique divided by the execution time of the optimal algorithm. The cost and execution time of the optimal technique will always be '1' and the amount of deviation of values from '1' will act as a performance measure of the GA and SA with respect to the optimal technique. Obviously the relative cost of GA or SA will be greater than or

equal to '1' and the relative time should be lesser than or equal to '1' (to be considered as useful alternative schemes).

Following specific values are used for the GA. *Pxover* = 0.85, *Pmutate* = 0.05, population size = 10. Stopping criterion is selected as no improvement in solution quality for 10 consecutive generations. Moreover, following specific values are used for the SA. Initial temperature (β_0) = 50, final temperature (β_f) = 0.01, cooling rate for temperature (α) = 0.8 (new temperature = 0.8 * current temperature)

Figure 13 shows the relative cost and relative execution time for Min-Sum objective function when P is equal to 2. There are a total of 6 graphs. Each pair of graphs shows, for GA and SA techniques, the relative cost and relative execution time, respectively. The three pair of graphs corresponds to the cases when the input graphs are of densities of *O(1)*, *O(logN)* and *O(N)*. We can make the following important observations. As per expectation the relative cost of GA or SA is greater than or equal to that of optimal technique while the relative time is lesser than or equal to that of optimal technique. Also, in general, as compared to the optimal technique the reduction in the execution time is far more than the degradation in the cost. For example, for *O(1)* graphs, a relative cost of 1.2 is obtained within a relative time of 0.3. In words a 20% increase in cost by reducing the execution time by 70% (actually in 80% of the cases the reduction in execution time is 95%). This suggests the use of GA or SA for the cases where execution time is a critical factor, without compromising much on the cost. Also from graphs it appears that SA is much faster than the GA (in almost all of the cases), while the solution quality too is better than GA in most of the cases. Note that in few cases the execution time of SA is very small as compared to the optimal thus making the relative time almost zero.

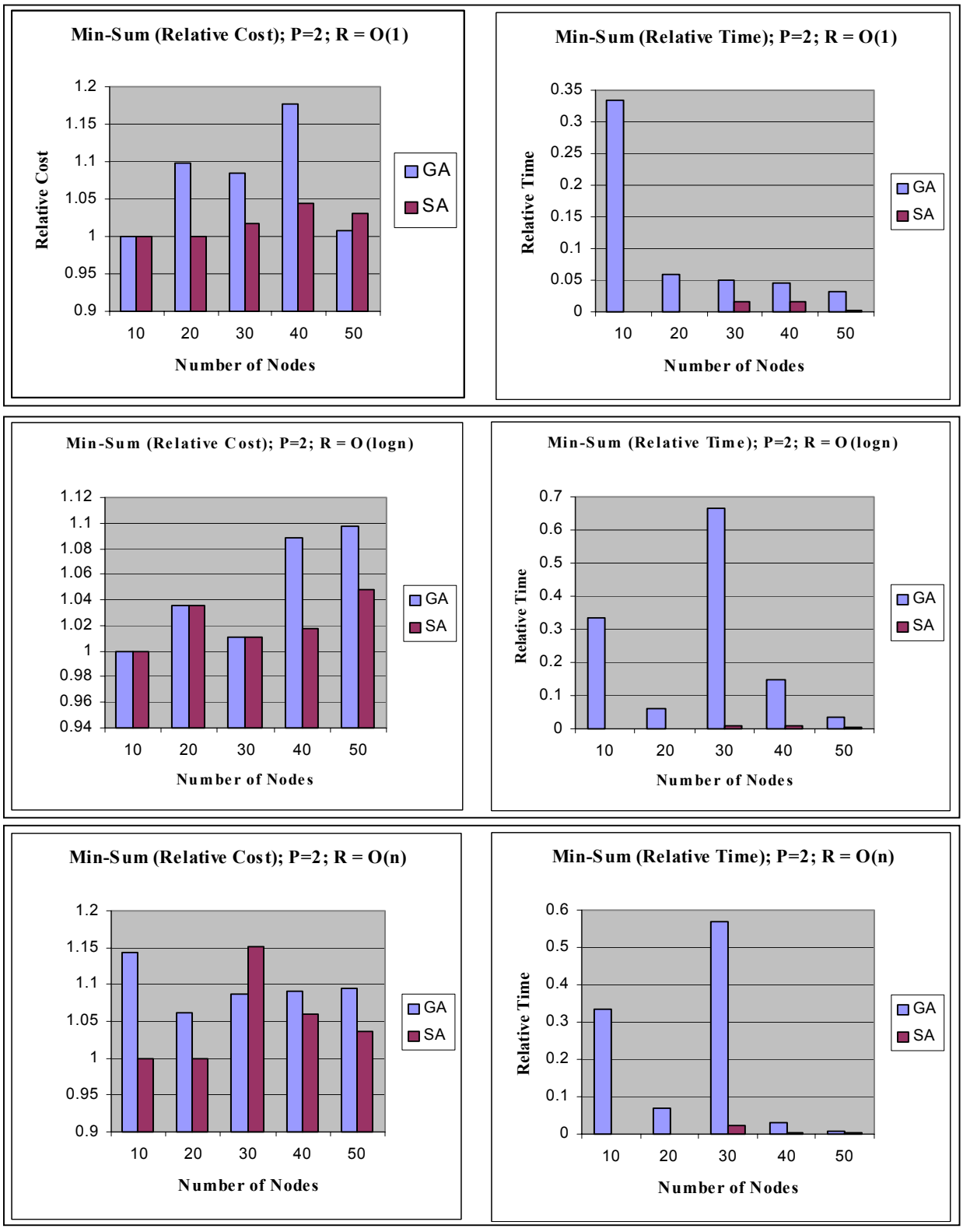

Figure 13: Relative cost and relative execution time for Min-Sum objective with graph densities of O(1), O(logn) and O(n), when P = 2.

Figure 14 shows the relative cost and relative execution time for Min-Sum objective function when P = 5. Again there are a total of six graphs; each pair of graphs corresponds to the scenario when input graphs were of densities of O(1), O(logn) and O(n). All the observations made for P=2, hold true when we increase P to 5.

Figure 15 shows the relative cost and relative execution time for Min-Max objective function with input graph densities of O(1), O(logn) and O(n), when P = 2. The above mentioned observations are valid with few interesting exceptions. For input graphs of O(1), SA performs poor than GA especially when there are less number of nodes. However, for larger number of nodes SA outperforms GA. This could be attributed to the population size in a generation for GA. Remember we are using a population size of 10. For small number of nodes, the parallel search of GA has an edge over the sequential search of SA. However, for larger number of nodes, SA manages to supersede GA owing to its better algorithm (e.g., its solution acceptance criteria and temperature schedule). Another worth noting case is when input graphs are *O(n)*. Both GA and SA manage to find out the optimal solution for all input cases, yet in a far less time.

Figure 16 shows the relative cost and relative execution time for Min-Max objective with graph densities of O(1), O(logn) and O(n), when P = 5. So far SA is out performing GA in terms of execution time, and in general is even performing better from the cost point of view. However, in figure 16, for O(n) scenario GA is equal to SA for most of the input cases (except when number of input nodes is equal to 40).

Note that in most of the cases GA shows a high variability in execution time. This can be attributed to its stopping criterion. Remember that we are stopping the execution when the solution does not improve after a specific consecutive number of generations. For different input parameters (and even for different runs) this could leads to different number of iterations

To summarize, we can conclude that both GA and SA are performing reasonably well in terms of cost when compared with the optimal technique, and are able to reduce the execution time in orders of magnitude. Also, SA is performing far better than GA both in terms of cost and the execution time.

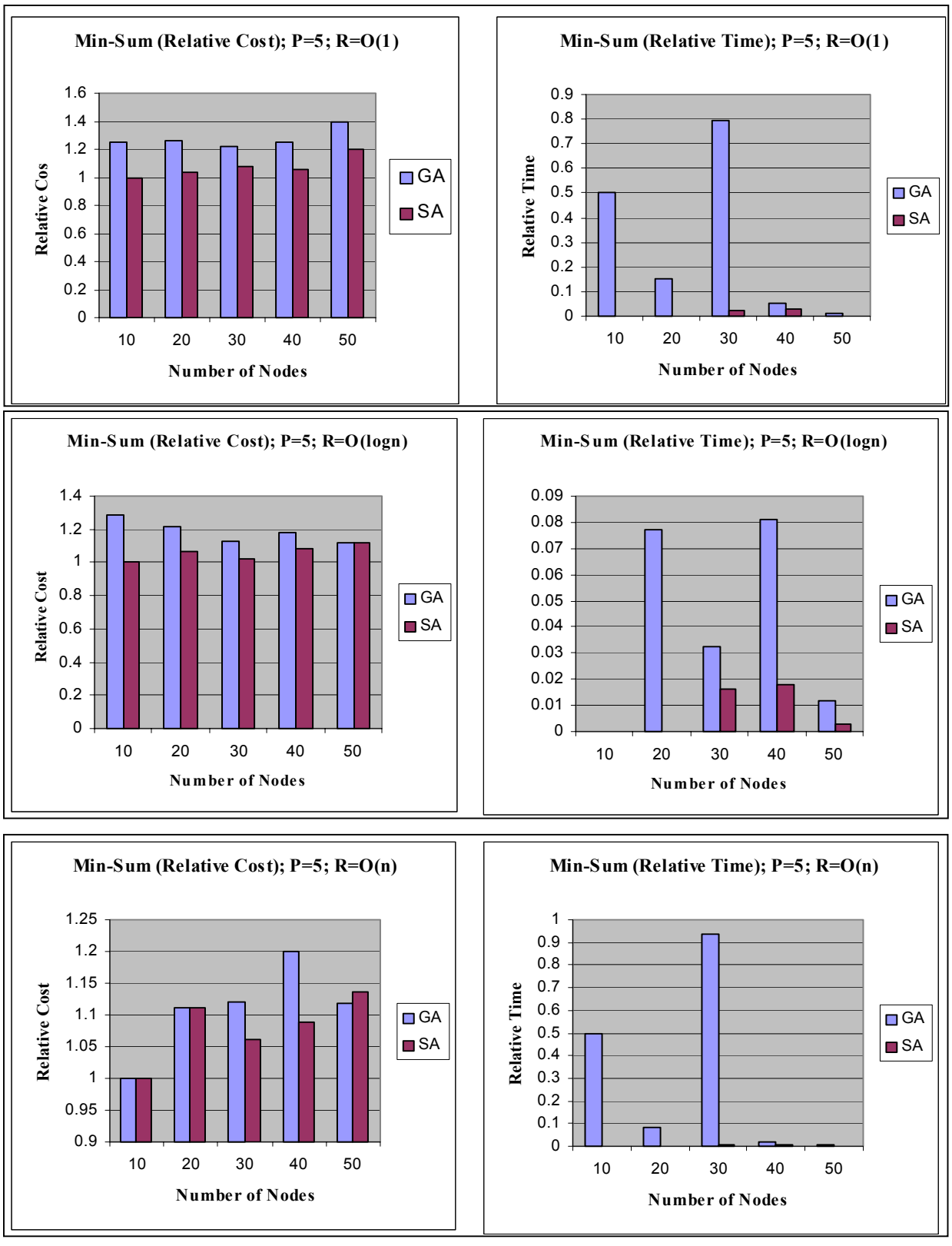

Figure 14: Relative cost and relative execution time for Min-Sum objective with graph densities of O(1), O(logn) and O(n), when P = 5.

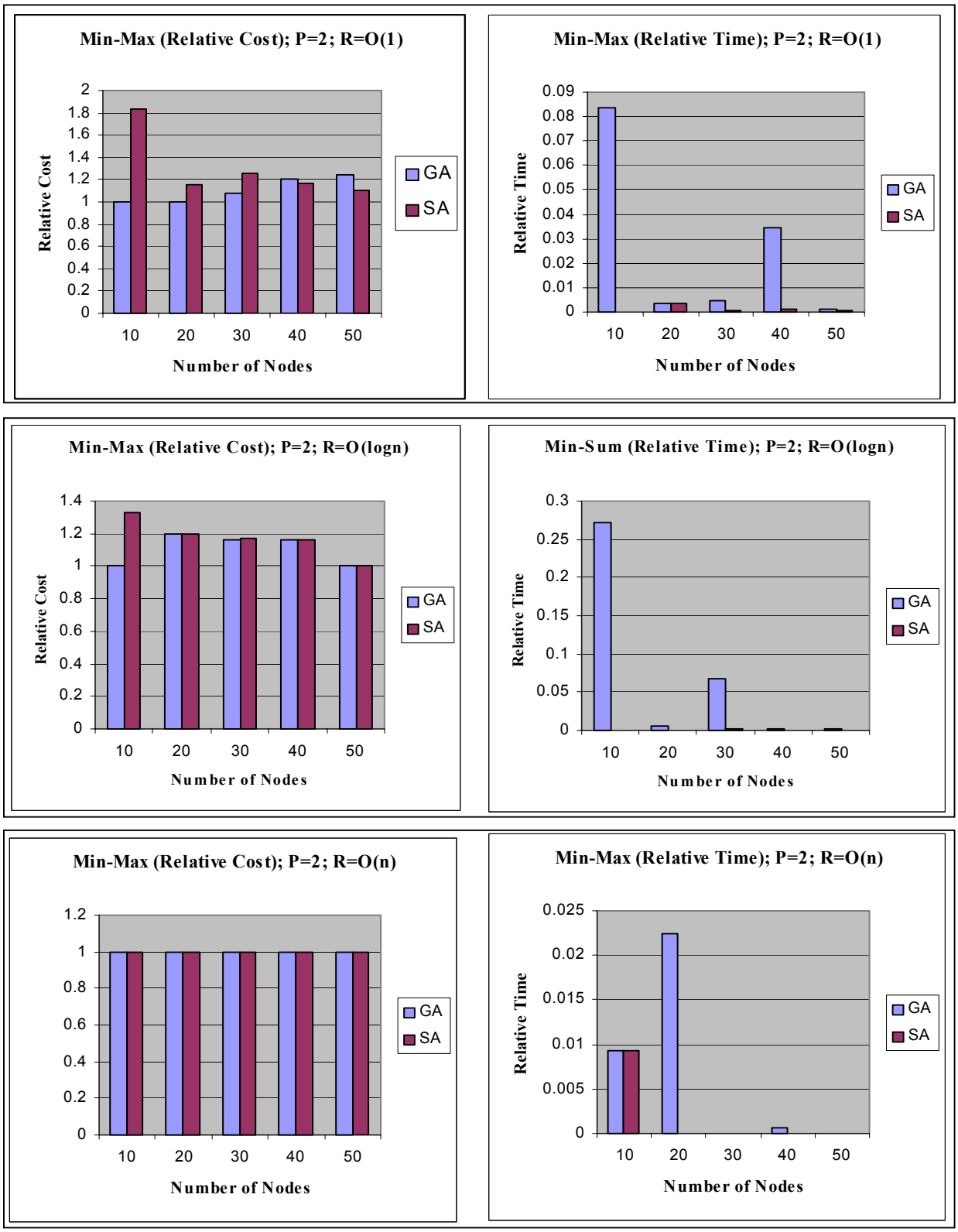

Figure 15: Relative cost and relative execution time for Min-Max objective with graph densities of O(1), O(logn) and O(n), when P = 2.

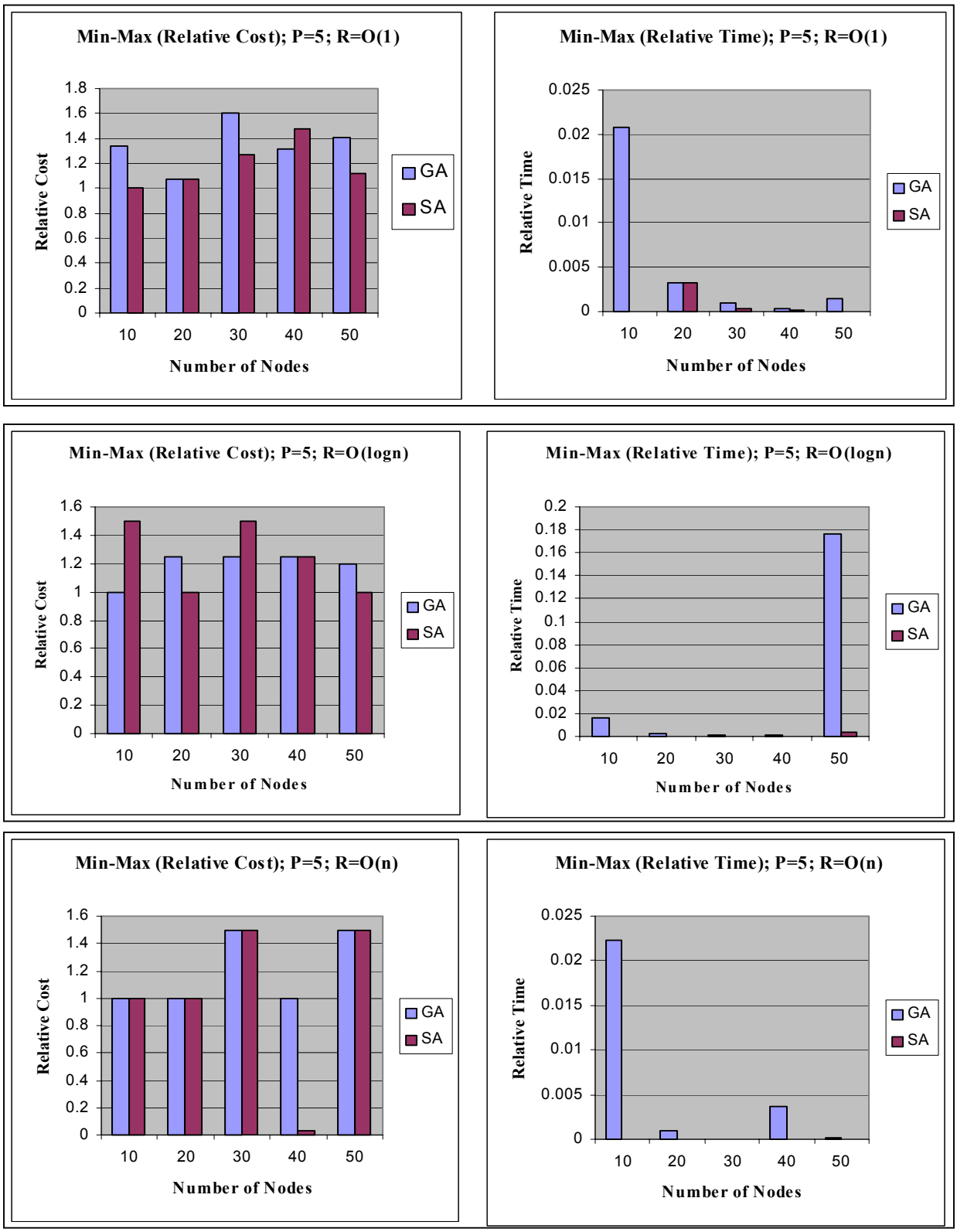

Figure 16: Relative cost and relative execution time for Min-Max objective with graph densities of O(1), O(logn) and O(n), when P = 5.

5.4 Network Design

Given a backbone graph, a network designer should be able to determine the number of proxy agents (p) that would result in an acceptable cost for both search and update operations. The search cost defined in Section 2 will typically decrease with the increase in p. This is because the increase in the number of proxy agents will result in smaller location areas with less search cost. On the other hand, the cost of update will increase as p increases. Plotting the search cost and the update cost should help network designers determine the best value of p.

In this experiment, we executed both the Min-Max and Min-Sum algorithms on several backbone graphs with a total of 50 nodes. We experimented with graph densities of $O(1)$, $O(\log n)$, and $O(n)$. In computing the update cost, we assumed that $k = 0.5$. We also assumed that the constants a and b equal 1 and 0, respectively. It should be noted that the choice of the coefficients should affect the selection of the minima. Normally these coefficients should reflect performance of the network. We plotted the search cost and update cost for different values of p in the range 1-50 as shown Figure 17.

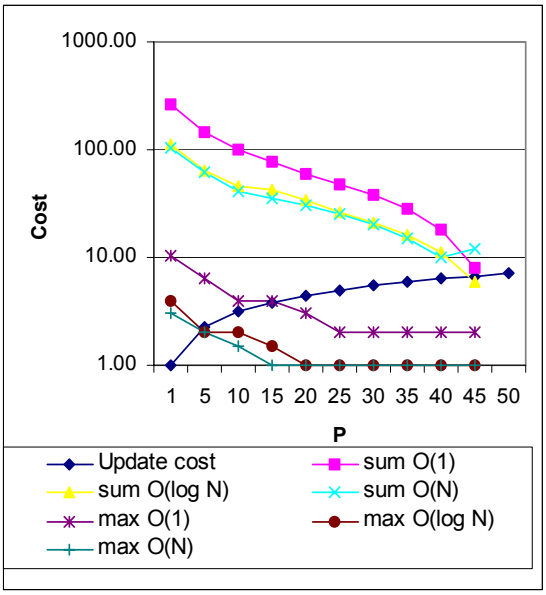

Figure 17: Network Design problem. Update Cost and corresponding Search Cost is plotted for different number of Proxy Agents. The search cost (in both cases Min-Sum and Min-Max) is decreasing with the increase in p while the update cost is increasing with the increase in p.

A network designer must select the value of p within the range of acceptable cost. Since several values of p can produce the same objective function, a designer can find the smallest value of p, which will not cause the cost to exceed a certain limit. The optimal points can be different depending on the whether we use the Min-Max or the Min-Sum criterion.

6. Conclusions

Our objective is to minimize the search cost that might result from reducing the number of update operations performed when a mobile host changes its subnetwork. We studied different approaches to identify a set of proxy agents that minimizes the cost of search. We used mathematical programming to obtain optimal solutions to the problem. We considered two situations: the cost of search is measured by the sum of all search message costs, and the cost of search messages depends on the maximum cost of such messages. For large networks in which the optimization problem may be intractable, we explored three different approximate approaches: 1) clustering, 2) genetic algorithms, and 3) simulated annealing.

Clearly the optimal search cost obtained took a lot more time than that when approximate approaches were applied. In small networks, optimal solutions can be obtained in reasonable time, while in large networks, approximate approaches must be used. Applying a simple clustering heuristic, that divides the backbone graphs into smaller sub-graphs, has shown promising results in producing not far from optimal solutions much faster. Also, both genetic algorithms and simulated annealing technique performed reasonably well in terms of cost when compared with the optimal technique, and were able to reduce the execution time in orders of magnitude. In addition, simulated annealing technique performed better than genetic algorithms both in terms of cost and the execution time.

Network designers can always select the number of proxy agents that will not cause the cost to fall beyond certain limits. The results reported in this paper have shown that this approach can be very useful in network design. For future work, we would like to conduct more experiments to accurately determine the network size for which the optimal algorithm can be used. Other clustering techniques will be explored for large networks. In addition, parallel execution of the algorithm will be investigated.

References

1. B. R. Badrinath, T. Imielinski and A. Virmani, "Locating Strategies for Personal Communication Networks," in *Proceedings IEEE Globecom 92 Workshop on Networking of Personal Communications Applications 1992*.
2. A. Bar-Noy and I. Kessler, "Tracking Mobile Users in Wireless Communications Networks," *IEEE Transactions on Information Theory*, Vol. 39, November 1993.
3. I. Chlamtac and A. Farago, "A New Approach to the Design and Analysis of Peer-to-Peer Mobile Networks", *Wireless Networks* 5 pp. 149-156, 1999.
4. B. Das and V. Bharagavan, "Routing in ad-hoc networks using minimum connected dominating sets," *IEEE International Conference on Communications (ICC'97)*, June 1997.
5. M. Handley et al., "SIP: Session Initiation Protocol," *Request for Comments 2543*, Internet Engineering Task Force, March 1999.
6. A. Kamal and H. El-Rewini, "On the Optimal Selection of Proxy Agents in Mobile Network Backbones," *Proceeding ICPP-2001*, pp. 103-110, September 2001.
7. M. Karaata and H. El-Rewini, "On Minimizing the Cost of Locating Users in Mobile Environments," *Journal of Parallel and Distributed Computing*, 61, pp. 950-966, 2001.
8. B. Liang and Z. Haas, "Predictive Distance-Based Mobility Management for PCS Networks," in *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, New York, March 1999.
9. H. Omar, T. Saadawi and M. Lee, "Supporting Reduced Location Management Overhead and Fault Tolerance in Mobile-IP Systems," *IEEE International Symposium on Computers and Communications*, 1999.
10. C. E. Perkins, "Mobile Networking in the Internet," *Mobile Networks and Applications*, Vol. 3, pp. 319-334, 1998.
11. C. E. Perkins, "Mobile Networking through Mobile IP," *IEEE Internet Computing Magazine* 2(1), pp. 58-69, January 1998.
12. C. E. Perkins, *Mobile IP - Design Principles and Practices*, Addison Wesley, 1998.

13. C. E. Perkins, ed., *IP mobility support, RFC 2002*, October 1996.

14. D. Wisely, P. Eardley and L. Burness, "IP for 3G: Networking Technologies for Mobile Communications", John Wiley and Sons, Chichester, U.K., 2002.

15. S.Rajgopalan and B. R. Badrinath, "Adaptive Location Management for Mobile-IP," *1st ACM International Conference on Mobile Computing and Networking - Mobicom'95*, November 1995.

16. I.F. Tsai, R.H. Jan, "The lookahead strategy for distance-based location tracking in wireless cellular networks," *MCCR* 3, 4, pp. 27-38.

17. J. Wu and H. Li, "On Calculating Connected Dominating Set for Efficient Routing in Ad Hoc Wireless Networks," in the *Proceedings of the Third International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications,* August 1999.

18. D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Edison Wesley, 1989.

19. S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing," *Science*, Vol. 220, pp. 671-680, 1983.